Coverage-Guided USB Fuzzing with Syzkaller

Andrey Konovalov <andreyknvl@google.com>

OffensiveCon
February 15th 2019

Who am I?

- Andrey Konovalov <andreyknvl@google.com>
- Software engineer at Google
- Work on various Linux kernel bug finding tools

- Twitter: @andreyknvl
- GitHub: @xairy
- Telegram: @xa1ry

Dynamic Tools Team

- Userspace
  - AddressSanitizer, ThreadSanitizer, MemorySanitizer, … (bug detectors)
  - libFuzzer (in-process coverage-guided fuzzing engine for C/C++)
  - oss-fuzz (continuous fuzzing service for open source software)

- Linux kernel
  - KASAN (available upstream), KMSAN (in active dev), KTSAN (on hold)
  - syzkaller (coverage-guided grammar-based kernel fuzzer)

Agenda

- Syzkaller overview
- USB overview
- USB fuzzing with syzkaller
- Hardware reproducers
- Bonus

Syzkaller Overview

Kernel Fuzzers Before Syzkaller

Trinity (and others) in essence:

```
while (true) syscall(rand(), rand(), rand());
```

Knows argument types, so more like:

```
while (true) syscall(rand(), rand_fd(), rand_addr());
```

- Tend to find shallow bugs
- Frequently no reproducers

Syzkaller

- Coverage-guided grammar-based kernel fuzzer
- Unsupervised
- Multi-
  - OS (Linux, *BSD, Fuchsia, …)
  - arch (x86-64, arm64, …)
  - machine (QEMU, GCE, Android phones, ...)
- Generates C reproducers for found bugs

Syzkaller

- As of now found over 2500 bugs (syzkaller/wiki/Found-Bugs + syzbot + internal)
- Numerous CVEs
- At least 4 public local privilege escalation exploits (CVE-2017-7308, CVE-2017-6074, CVE-2017-2636, CVE-2017-1000112)

Coverage for the Linux Kernel

- Available upstream with CONFIG_KCOV
- GCC/Clang pass that inserts a function call into every basic block
- Kernel debugfs extension that collects and exposes coverage per-thread

```
if (...) {
    ...
}
```

```
__sanitizer_cov_trace_pc();  // 1
if (...) {
     __sanitizer_cov_trace_pc();  // 2
    ...
}
__sanitizer_cov_trace_pc();  // 3
```

Syscall Descriptions

Declarative description of all syscalls:

```
open(file filename, flags flags[open_flags], mode flags[open_mode]) fd
read(fd fd, buf buffer[out], count len[buf])
close(fd fd)

open_flags = O_RDONLY, O_WRONLY, O_RDWR, O_APPEND ...
```

Programs

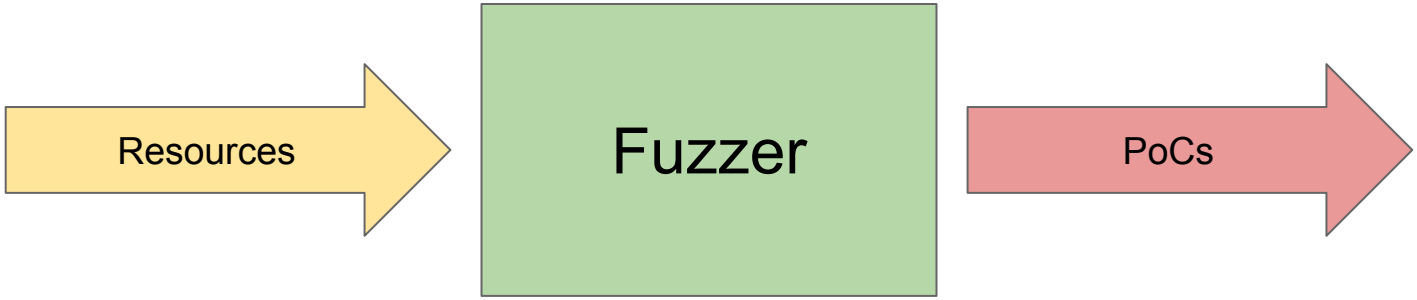
The description allows to **generate** and **mutate** "programs" in the following form:

```
mmap(&(0x7f0000000000), (0x1000), 0x3, 0x32, -1, 0)
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
read(r0, &(0x7f0000000000), 42)
close(r0)
```

Algorithm

1. Start with an empty corpus of programs
2. Generate a new program, or choose an existing program from corpus and mutate it
3. Run the program, collect coverage
4. If new code is covered, minimize the program and add to the corpus
5. Goto 1

Ideally...



Reality

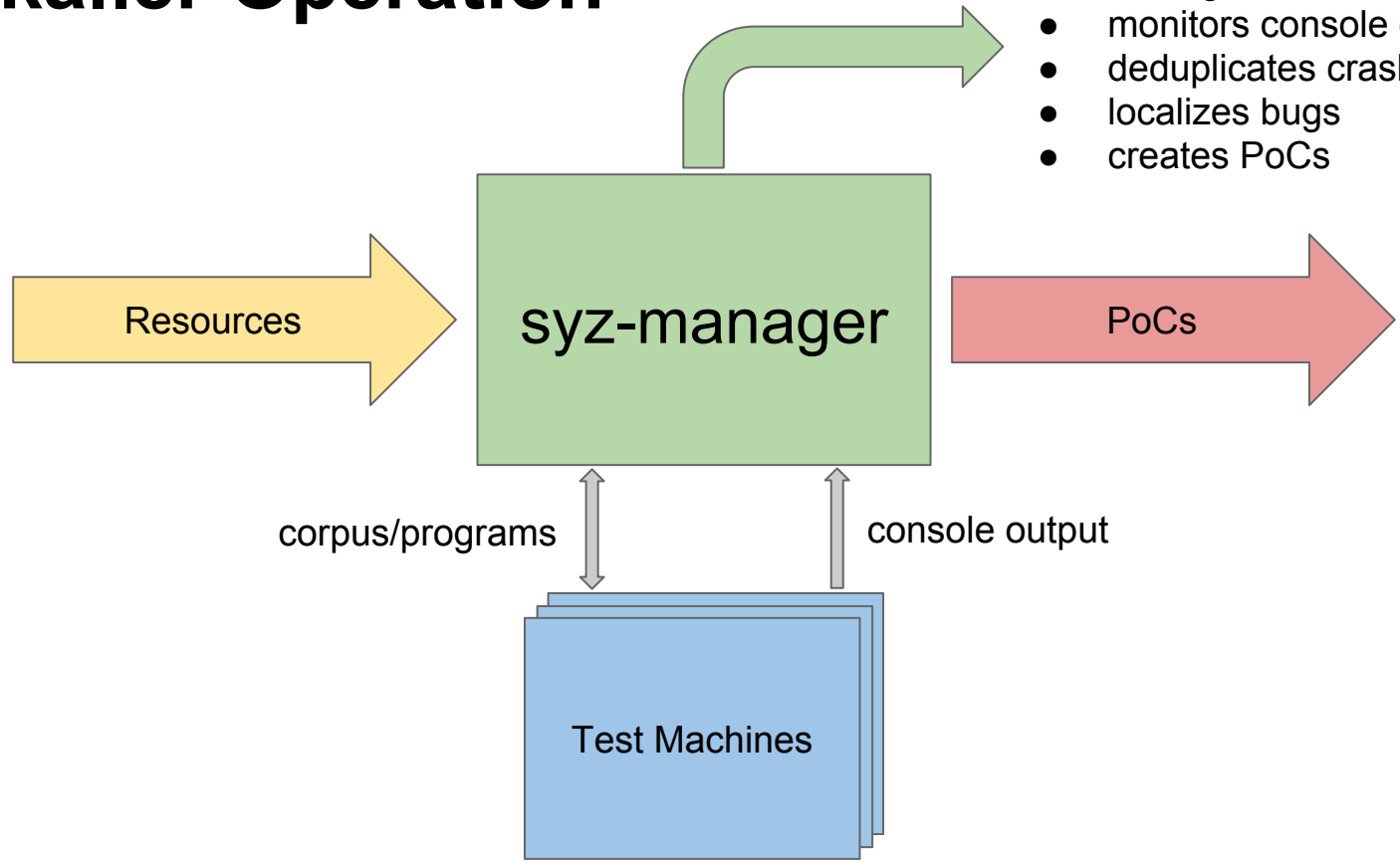
Operation of a typical kernel fuzzer:

- Manually create a bunch of VMs
- Manually copy and start the binary
- Manually monitor console output
- Manually deduplicate crashes
- Manually localize and reproduce
- Manually restart the crashed VM (or press power button!)

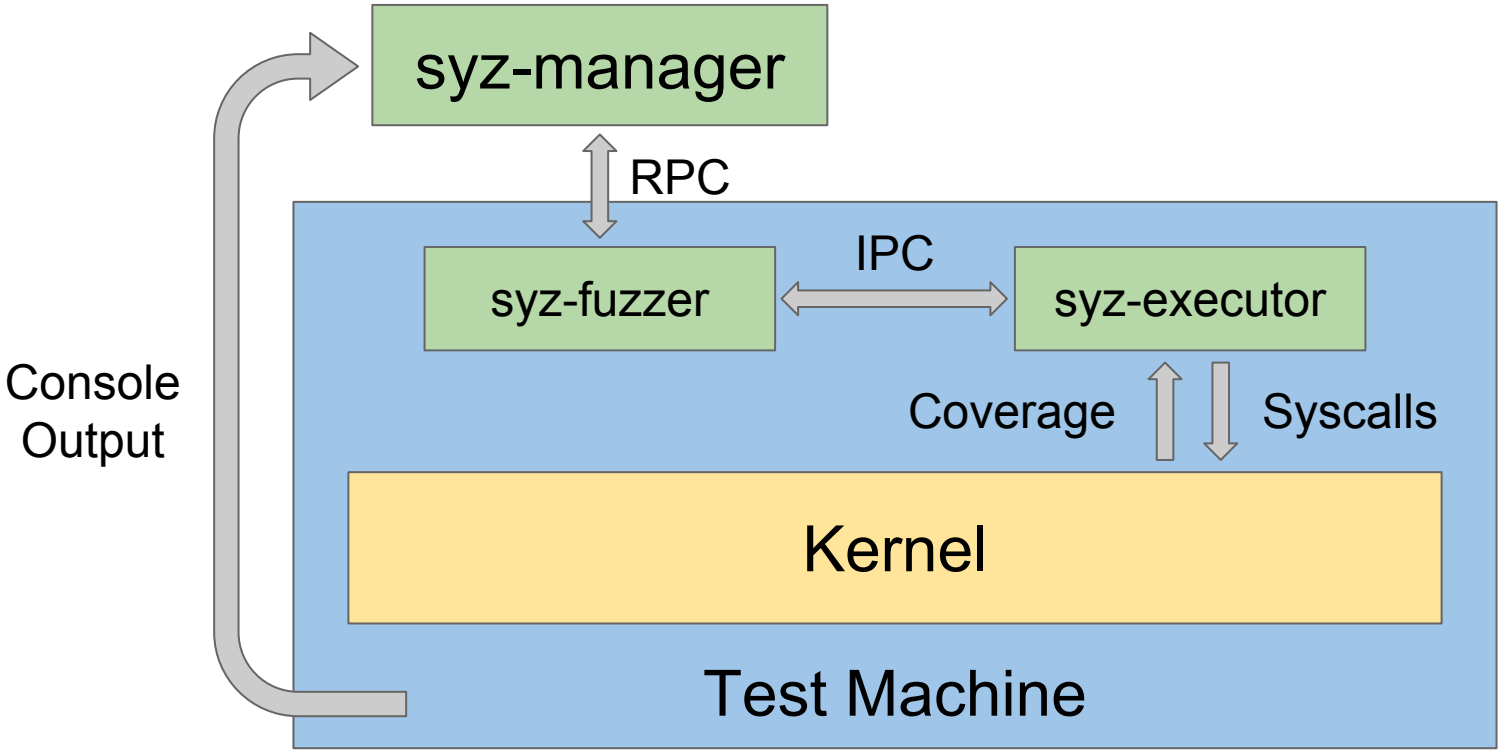
Works only if you want to find a handful of crashes. Otherwise - full time job.

Syzkaller Operation

- manages test machines
- monitors console output
- deduplicates crashes
- localizes bugs
- creates PoCs



Syzkaller Architecture



Further Automation

- syz-ci
  - Continuous kernel/syzkaller updates
  - Restarts syz-manager on new kernel/syzkaller
- syz-hub
  - Input/reproducer exchange between several syz-manager's
- syzbot
  - Bug aggregation
  - Reporting
  - Status tracking

syzkaller.appspot.com

USB Overview

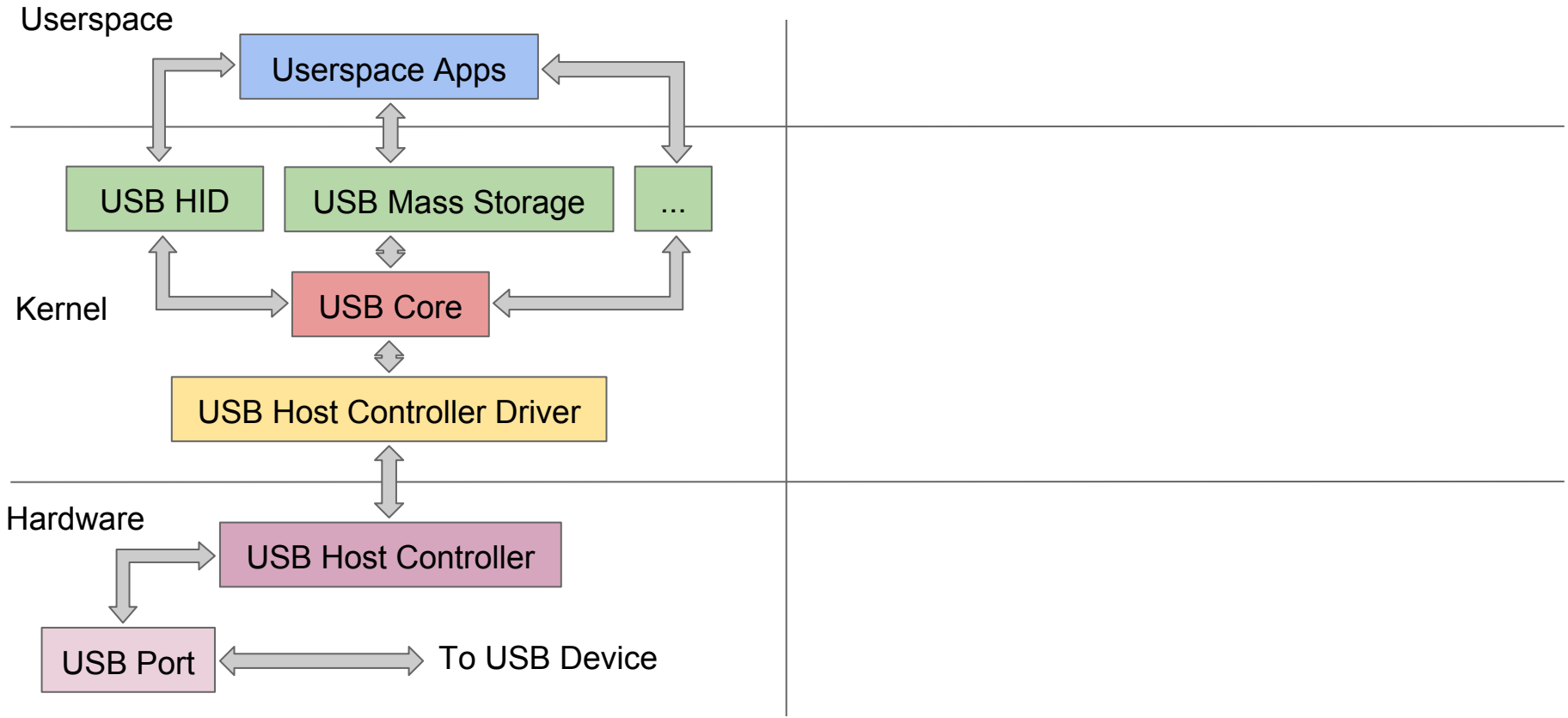
USB 101

- Host initiates all communication with a Gadget (a USB device)
- Message based protocol
- When a Gadget is connected, Host performs an enumeration procedure to find out which driver should handle the Gadget
- During enumeration Gadget sends USB descriptors that describe it to the Host
- After enumeration has completed, Host gives away Gadget handling to the loaded driver
- Good read on the USB protocol basics: USB 101: An Introduction to Universal Serial Bus 2.0 by Robert Murphy

Demo: lsusb

Demo: usbmon

Linux USB Subsystem: Host



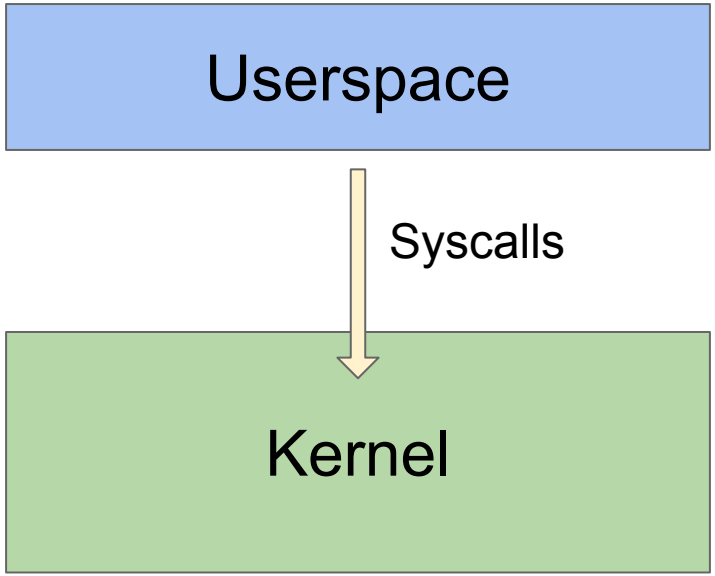
Previous USB Fuzzing Approaches

- Use hardware
  - FaceDancer - "The purpose of this board is to allow USB devices to be written in host-side Python, so that one workstation can fuzz-test the USB device drivers of another host"

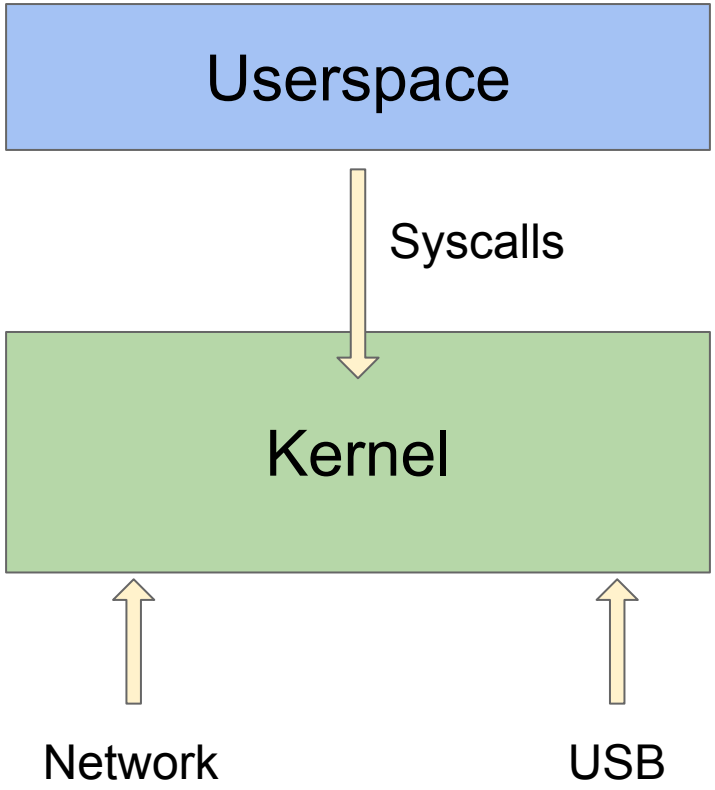
- Emulate USB devices though a hypervisor
  - vUSBf - fuzzes the guest kernel running in QEMU by connecting USB devices via usbredir protocol

USB Fuzzing with Syzkaller

Kernel Inputs: Syscalls



External Kernel Inputs



Goal: Fuzzing the USB Stack Externally

- We want to collect coverage from the USB stack
  - Currently CONFIG_KCOV allows to collect coverage from the current kernel thread, but USB is handled asynchronously in a background thread

- We want to deploy this on syzbot
  - Syzbot fuzzes the kernel on GCE VMs, thus we can't depend on some external entities injecting USB packets, like hypervisors (e.g. QEMU) or hardware (e.g. FaceDancer)

Collecting Coverage From USB

- CONFIG_KCOV allows to collect coverage from the current kernel thread
- USB handling happens in background kernel threads (hub_event)

- The idea is to annotate background kernel thread code and collect coverage (using the same compiler callbacks) into a per-annotation buffer, and then copy it to the userspace coverage memory
- Patches that extend CONFIG_KCOV to support coverage collection from arbitrary kernel threads (not upstream yet): [1], [2]

Collecting Coverage with CONFIG_KCOV

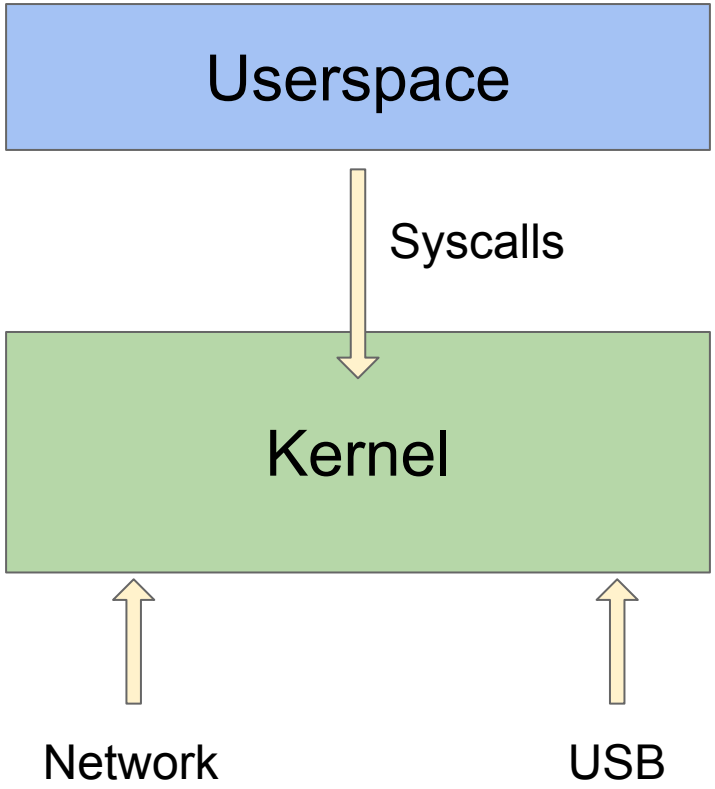
```
int fd = open("/sys/kernel/debug/kcov", ...);

unsigned long *cover = mmap(NULL, ..., fd, 0);

ioctl(fd, KCOV_ENABLE, ...);

// Now coverage from the current kernel thread is collected into cover.
```

Coverage From Background Threads

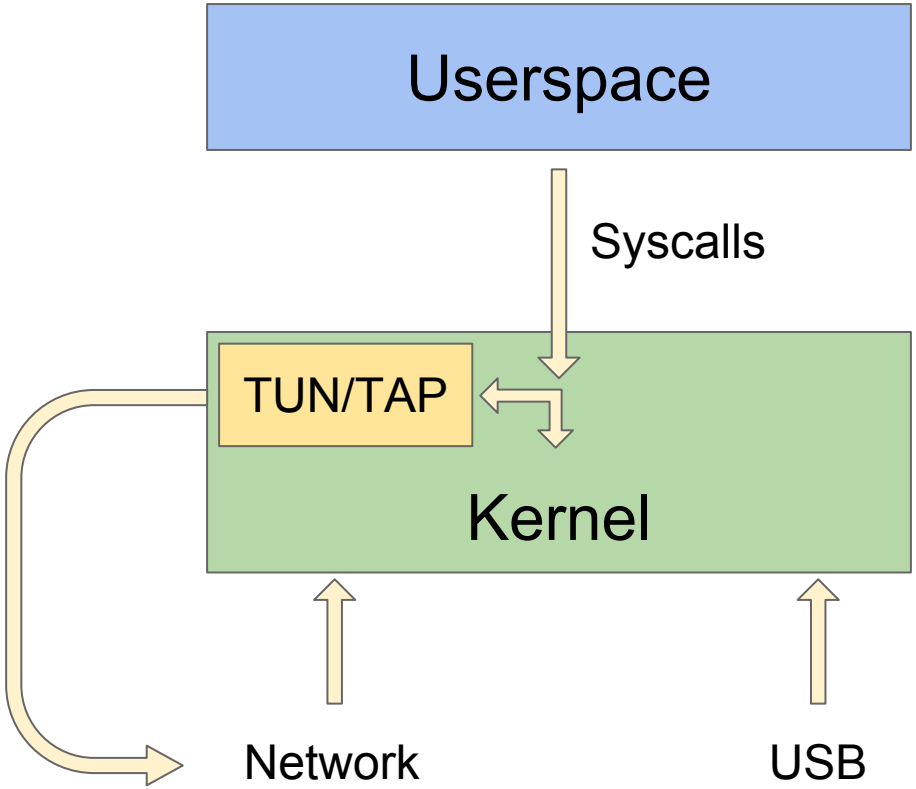
```
void hub_event() {  // Executed in background kernel thread.
    kcov_remote_start(UNIQUE_ID);  // Requires kernel annotations.
    ...
    kcov_remote_stop();  // This dumps collected coverage into *cover.
}
```

```
int fd = open("/sys/kernel/debug/kcov", ...);
unsigned long *cover = mmap(NULL, ..., fd, 0);
ioctl(fd, KCOV_REMOTE_ENABLE, {..., UNIQUE_ID, ...});
// Now coverage from background kernel thread is collected into cover.
```

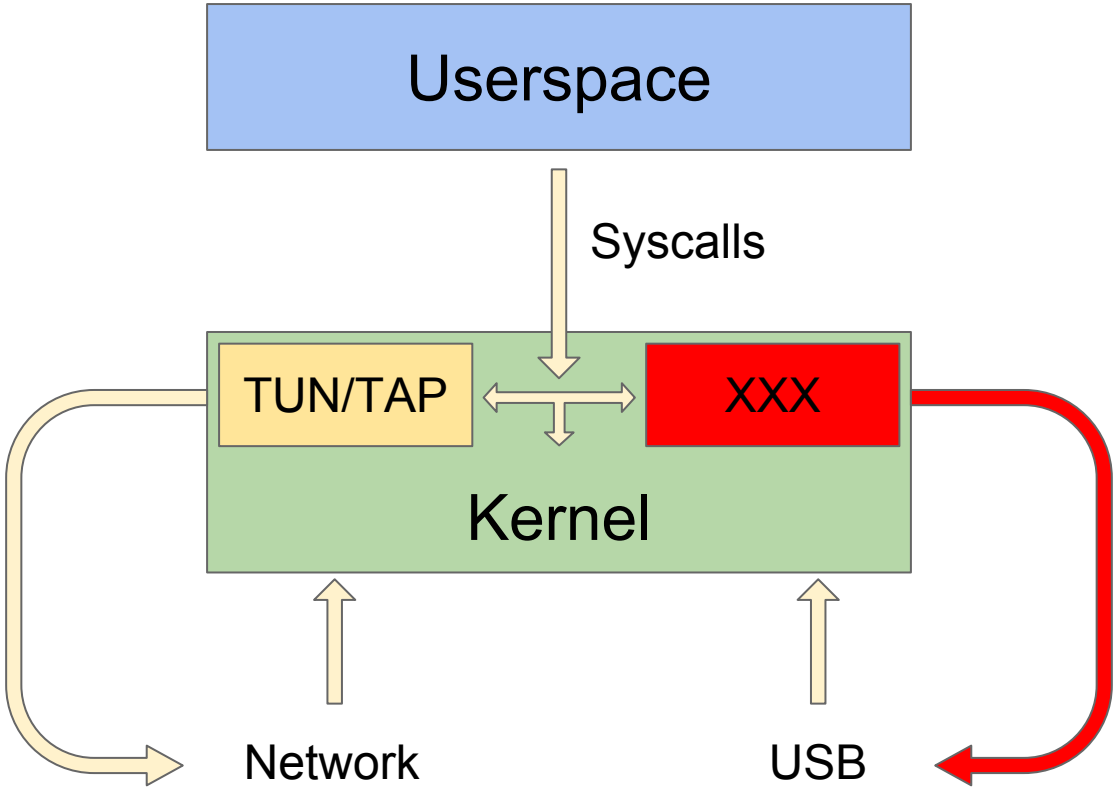
How Do We Inject USB "Packets"?



Syzkaller Injects Net Packets via TUN/TAP



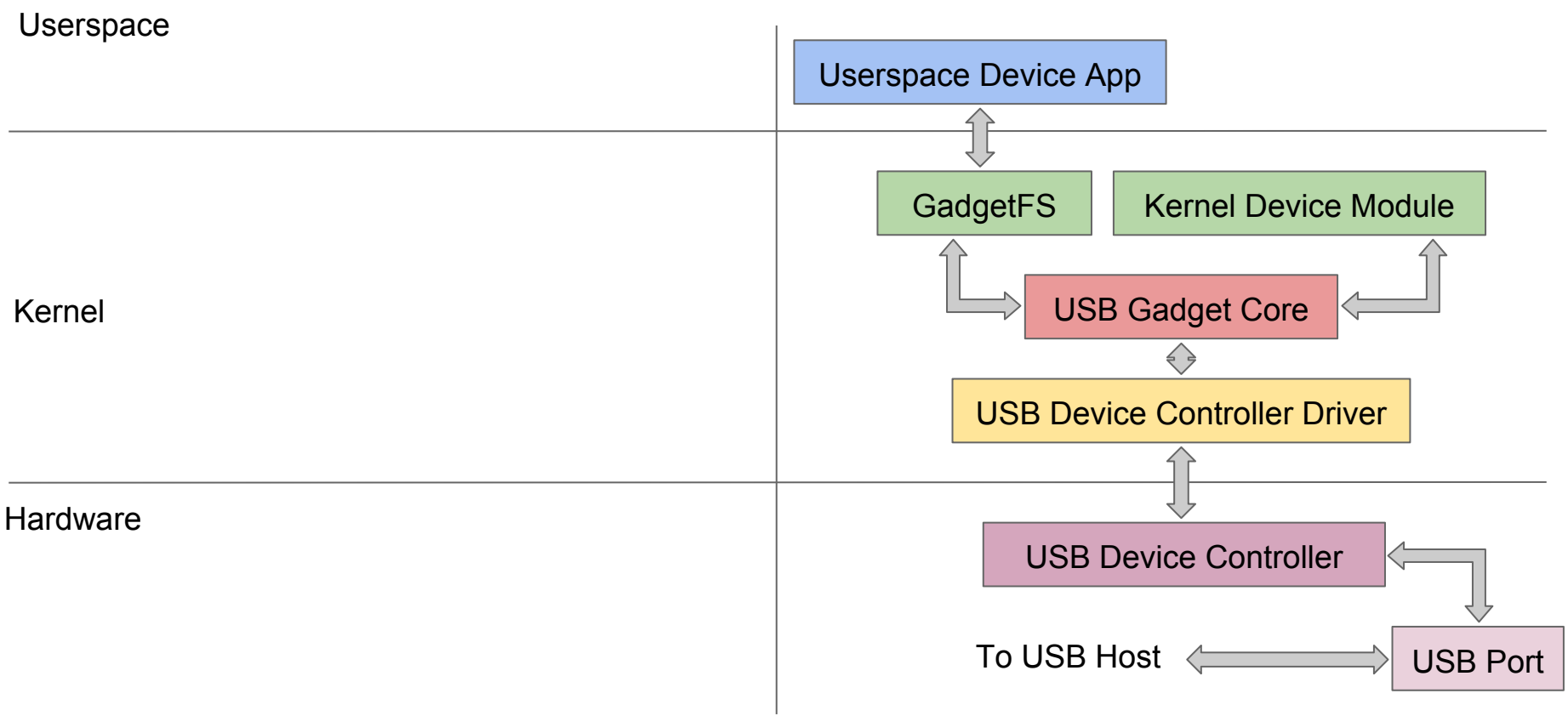
Can We Do Something Similar for USB?



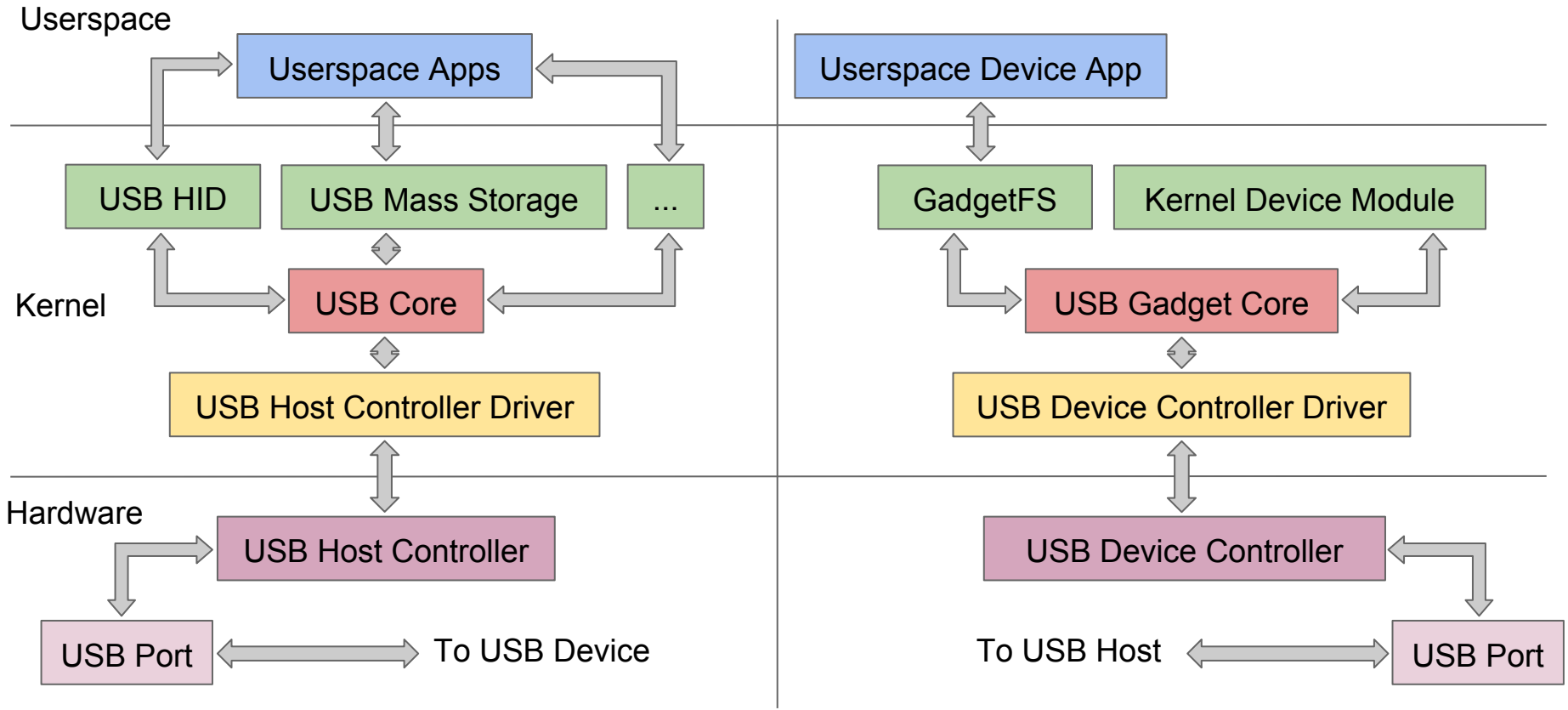
Linux USB Subsystem: Gadget

- Allows to turn a Linux device into a USB device
- Requires a USB Device Controller and driver for it
- Linux provides a few interfaces for the Gadget Subsystem
  - GadgetFS - a userspace interface, essentially allows to implement USB device logic as a userspace app
  - ConfigFS/FunctionFS and legacy modules (g_hid.ko, ...) - implement USB device logic as kernel modules, but are controlled from userspace
  - Gadget Subsystem API - allows to implement a USB device as a custom kernel module

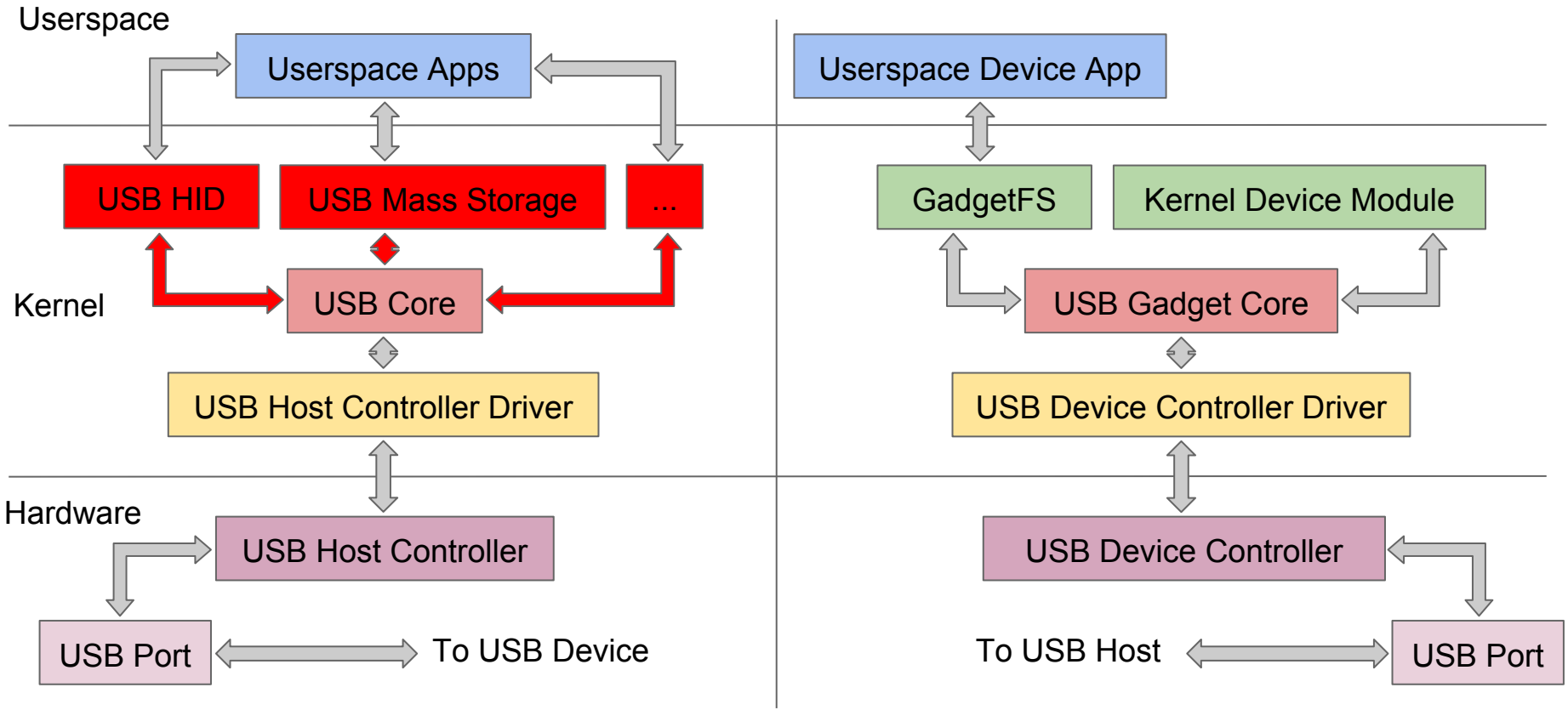
Linux USB Subsystem: Gadget



Linux USB Subsystem



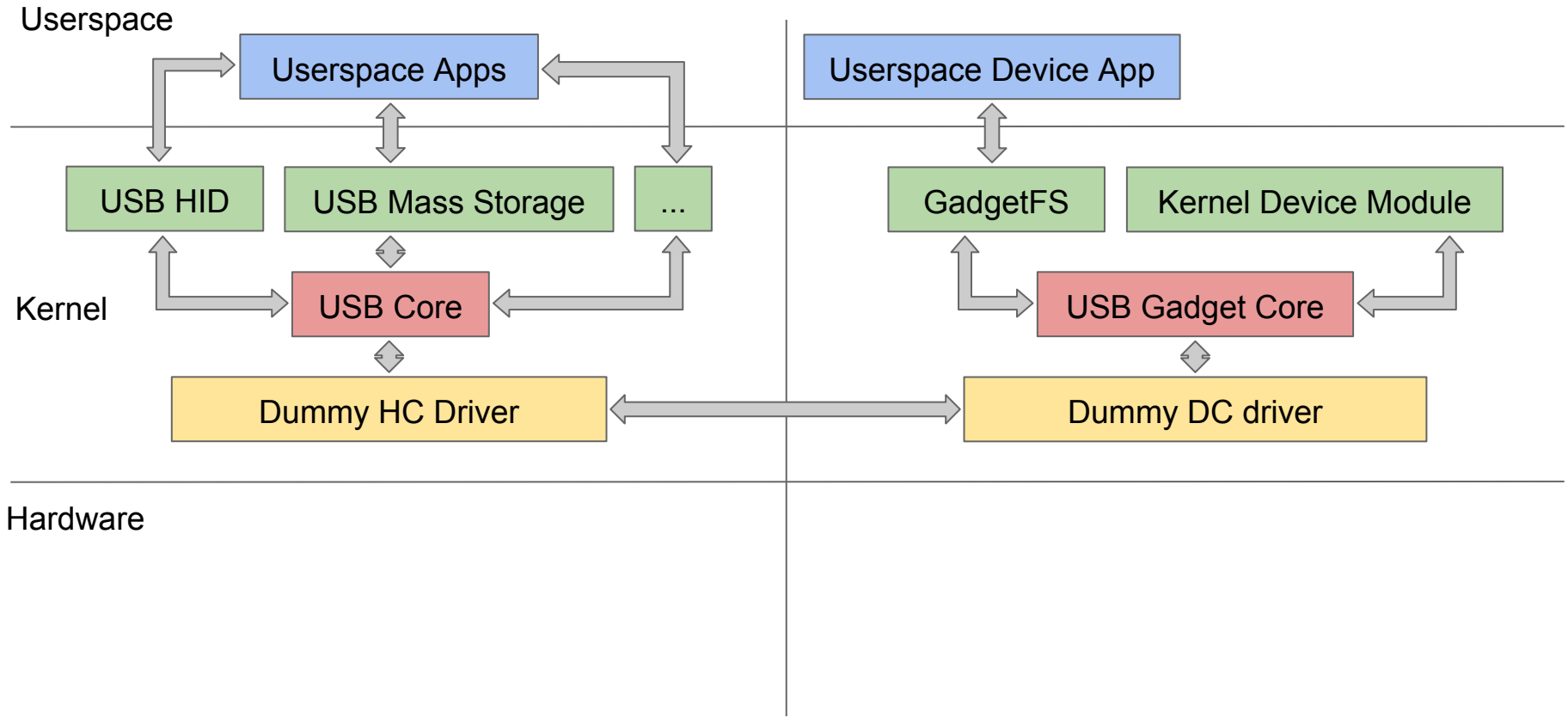
Linux USB Subsystem: Attack Target



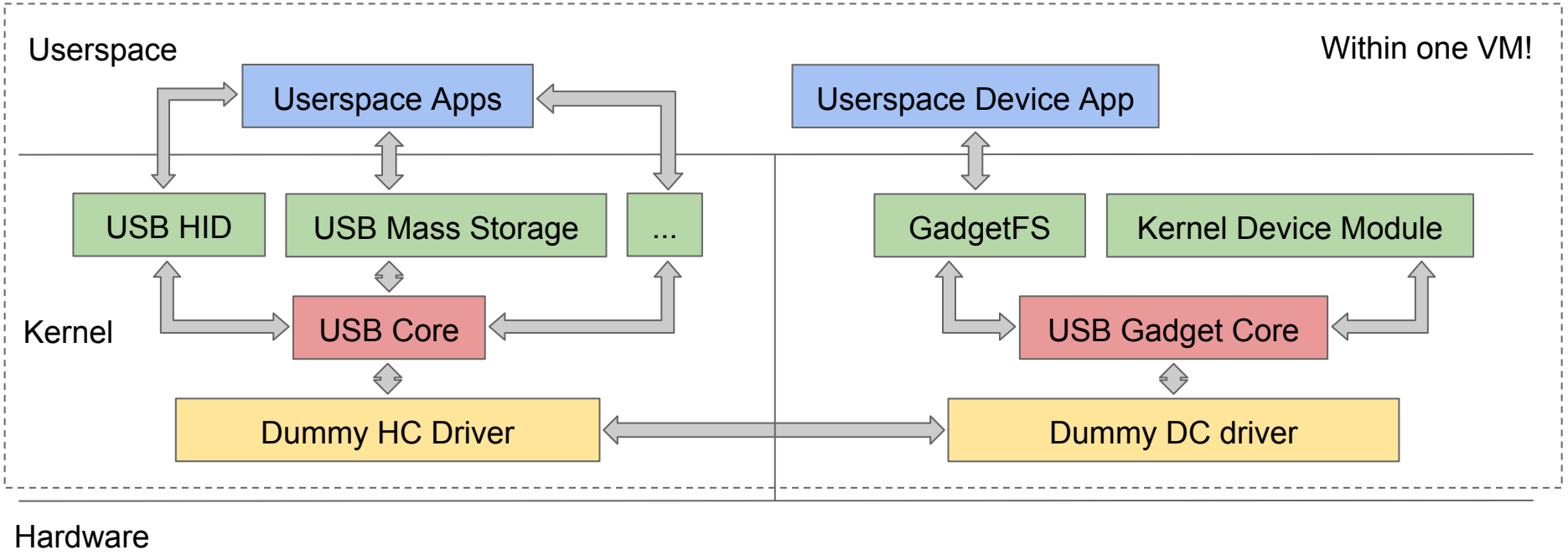
CONFIG_USB_DUMMY_HCD

- Implements virtual USB Host Controller Driver and USB Device Controller Driver that are connected within the kernel and don't require any hardware
- Essentially allows to emulate USB devices from userspace
- Available upstream

CONFIG_USB_DUMMY_HCD



CONFIG_USB_DUMMY_HCD

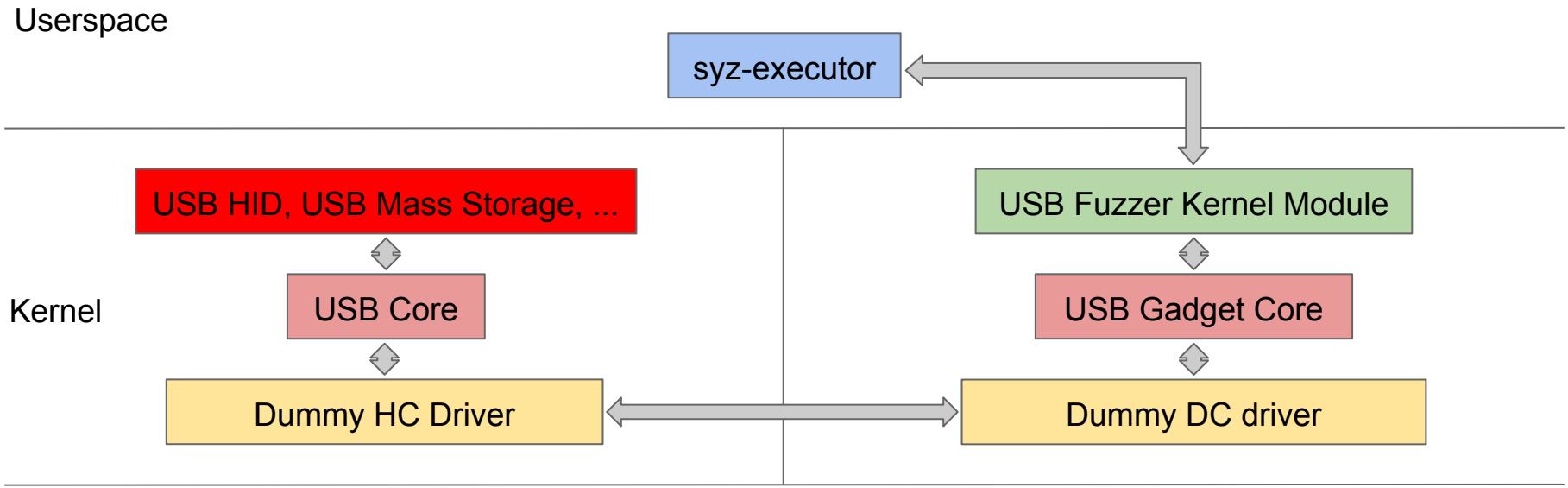


No hardware (or hypervisors) required!

USB Fuzzer Kernel Module

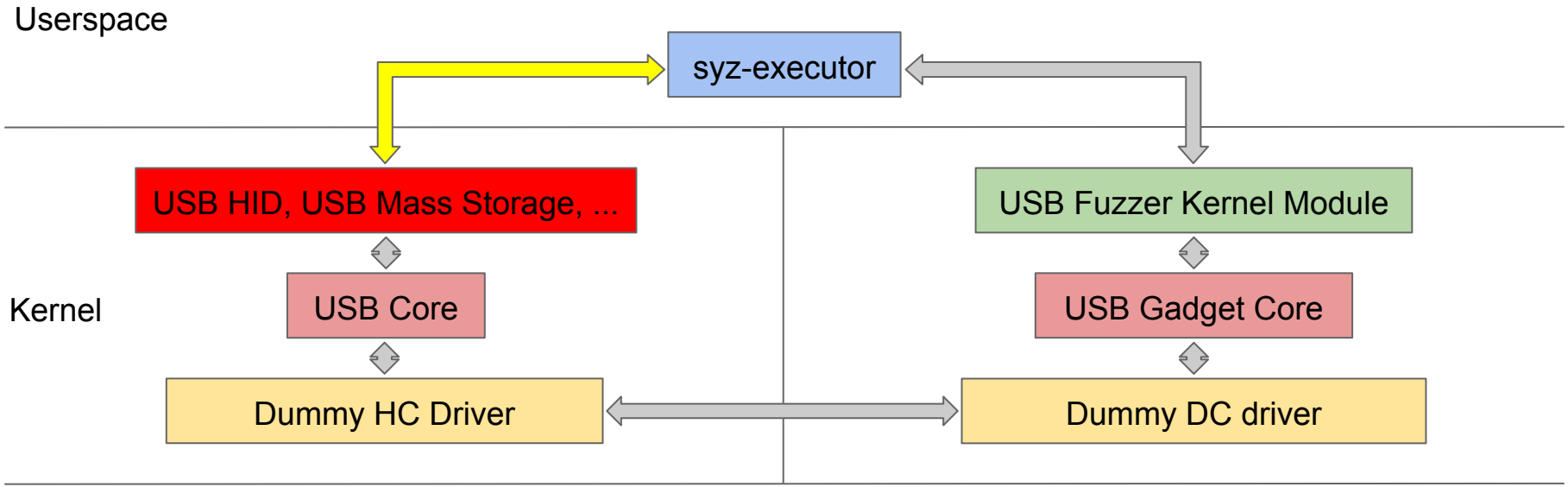
- A GadgetFS-like interface for emulating USB devices via the Linux Gadget Subsystem
- Uses Gadget API for communication with the USB Device Controller
- Provides an ioctl-based userspace interface for emulating USB devices
- Unlike GadgetFS doesn't have sanity checks on the provided USB descriptors and doesn't contain any device emulation logic
- All of the USB device emulation logic is controlled by the userspace
- Not yet upstream, the patch is here

Syzkaller USB Fuzzing Approach



No hardware (or hypervisors) required!

Syzkaller USB Fuzzing Approach (not yet)



No hardware (or hypervisors) required!

Syzkaller USB Descriptions

- sys/linux/vusb.txt contains the generic descriptions of USB messages and descriptors
- sys/linux/vusb_ids.txt contains an automatically extracted list of USB IDs that are used to match a USB device to a driver
- executor/common_linux.h contains the implementation of USB emulation routines (syz_usb_connect)

syz_usb_connect

```
syz_usb_connect(usb_device, usb_message_descriptions) {
    fd = open("/sys/kernel/debug/usb-fuzzer");
    connect_device(fd, usb_device);
    while (event = wait_for_usb_event(fd)) {
        response = find_matching_response(event):
        send_response(fd, response);
    }
}
```

Demo: USB Fuzzing In Progress

Demo: Running Reproducers in a VM

Linux Kernel USB Fuzzing Results

- Reported 80+ bugs in the USB subsystem
- 5 bugs in USB core subsystem
- 23 CVEs (for the bugs that got fixed)
- 50+ not yet reported, working on syzbot integration
- 12 bugs in USB Gadget Subsystem (mostly GadgetFS)

Limitations

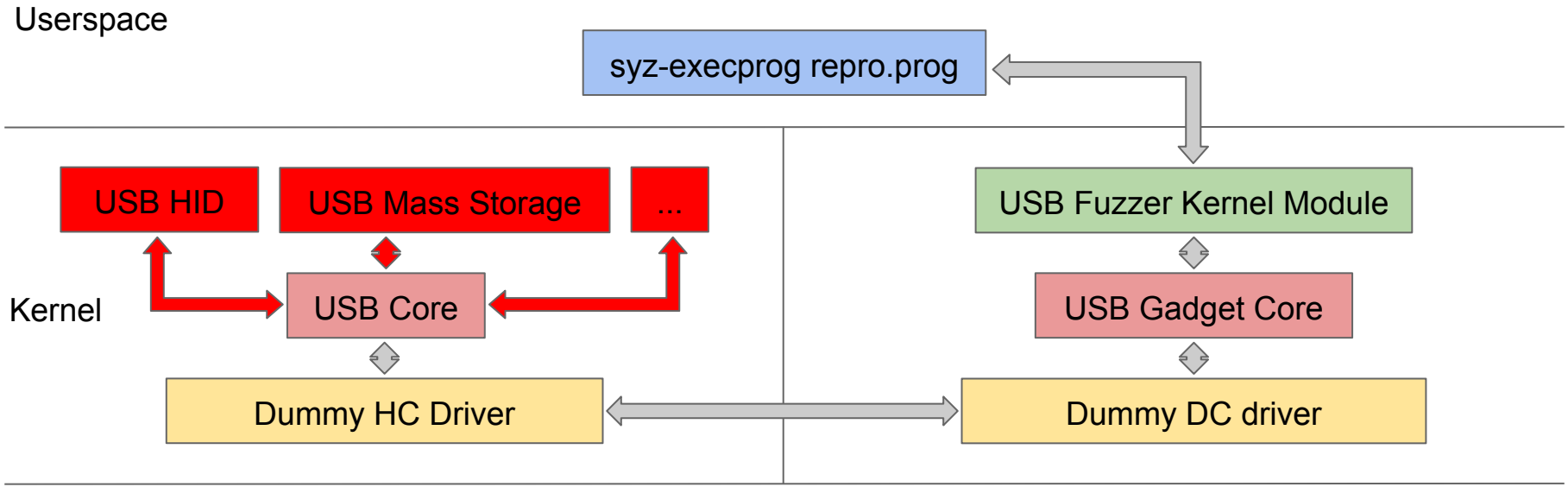
- Linux only
- CONFIG_USB_DUMMY_HCD doesn't support isochronous transfers
- Linux Gadget API doesn't expose some low level USB requests (e.g SET_FEATURE)

Future Work

- Improve fuzzing after enumeration (need coverage and descriptions for specific protocols)
- Upstream Linux kernel patches and syzkaller changes
- Integrate with syzbot for automatic bug reporting
- Fuzz other protocols besides USB

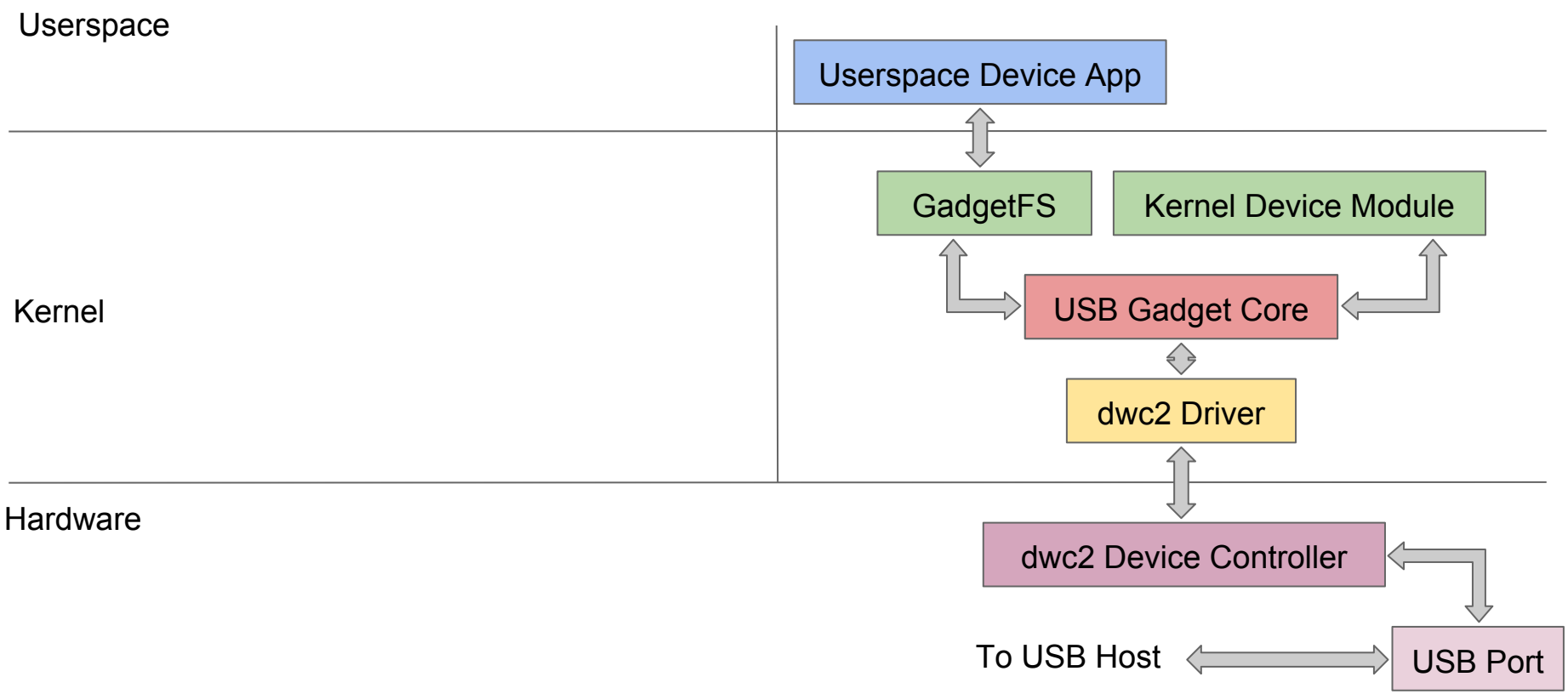
Hardware Reproducers

Running Reproducers via Dummy

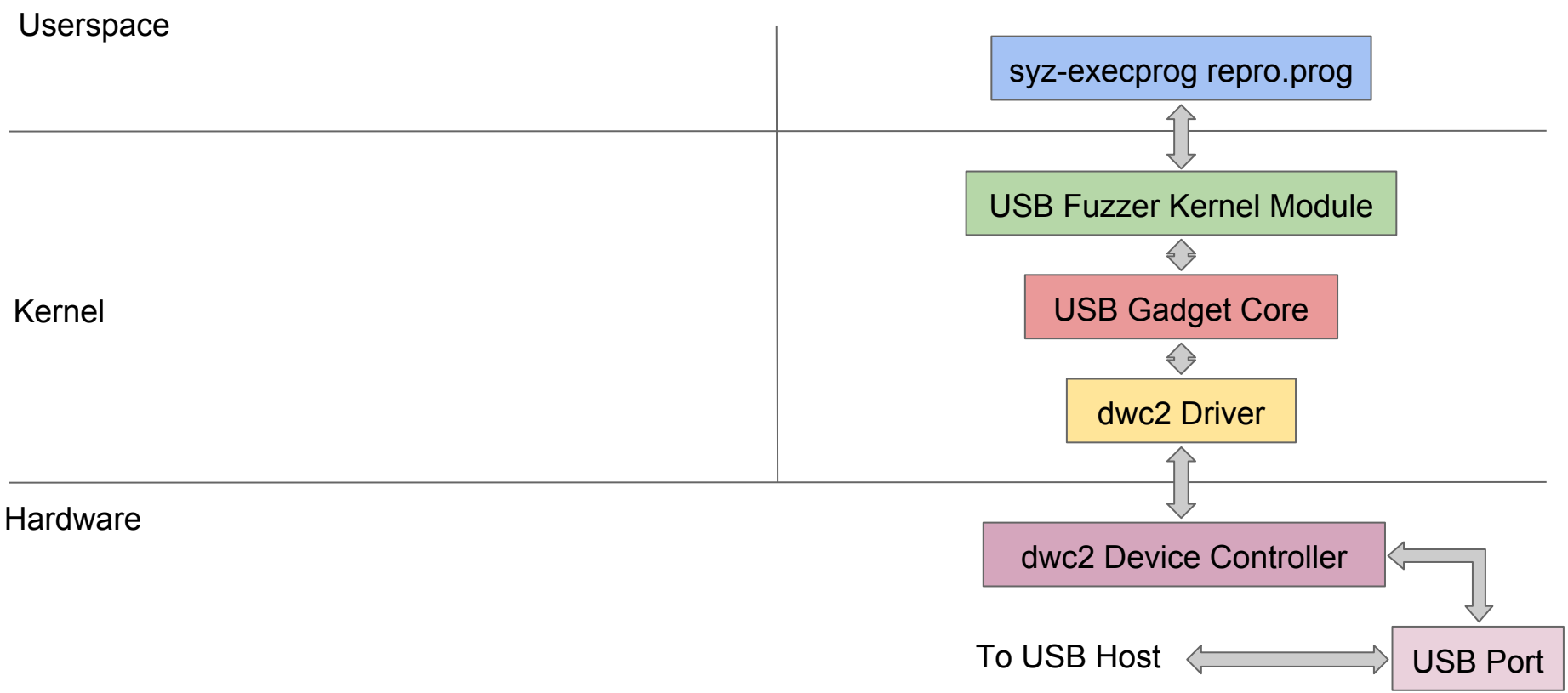


No hardware required!

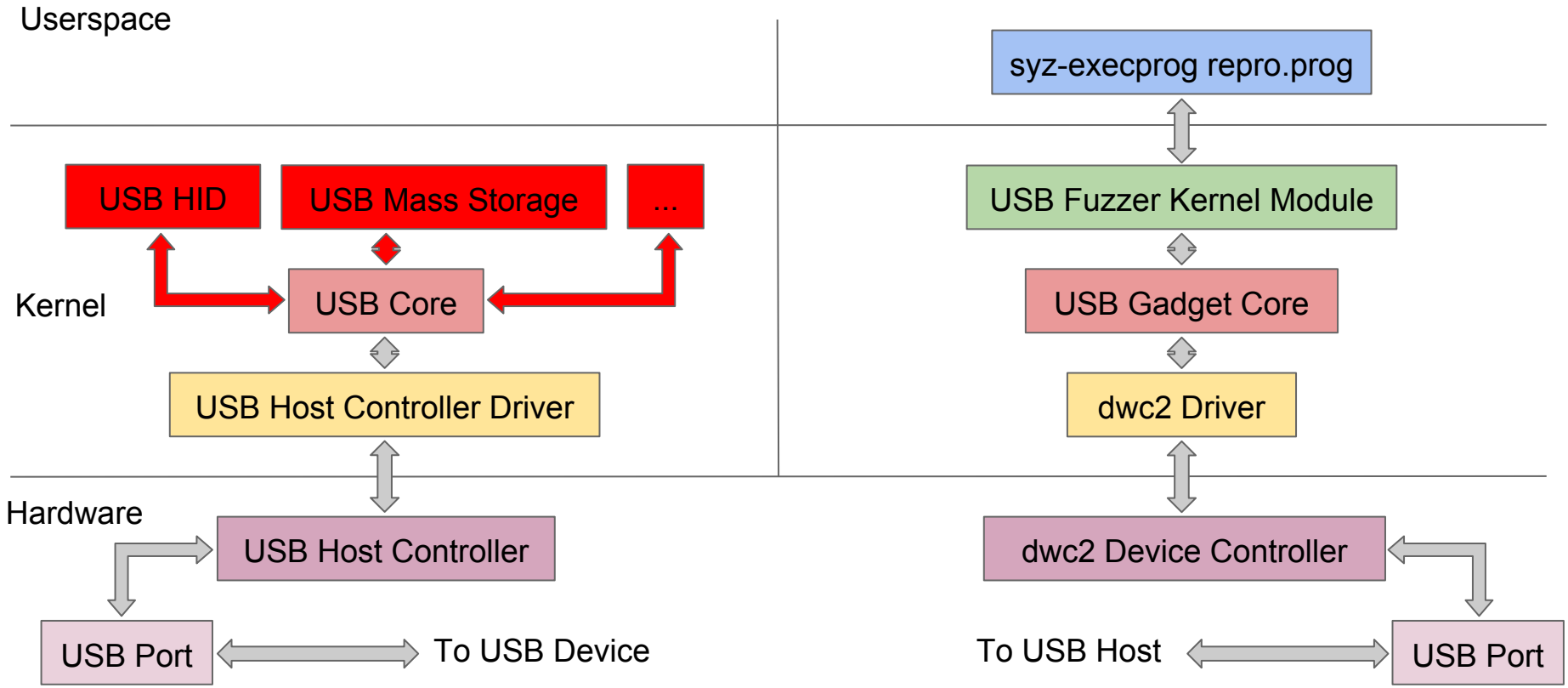
Raspberry Pi Zero: dwc2 Device Controller



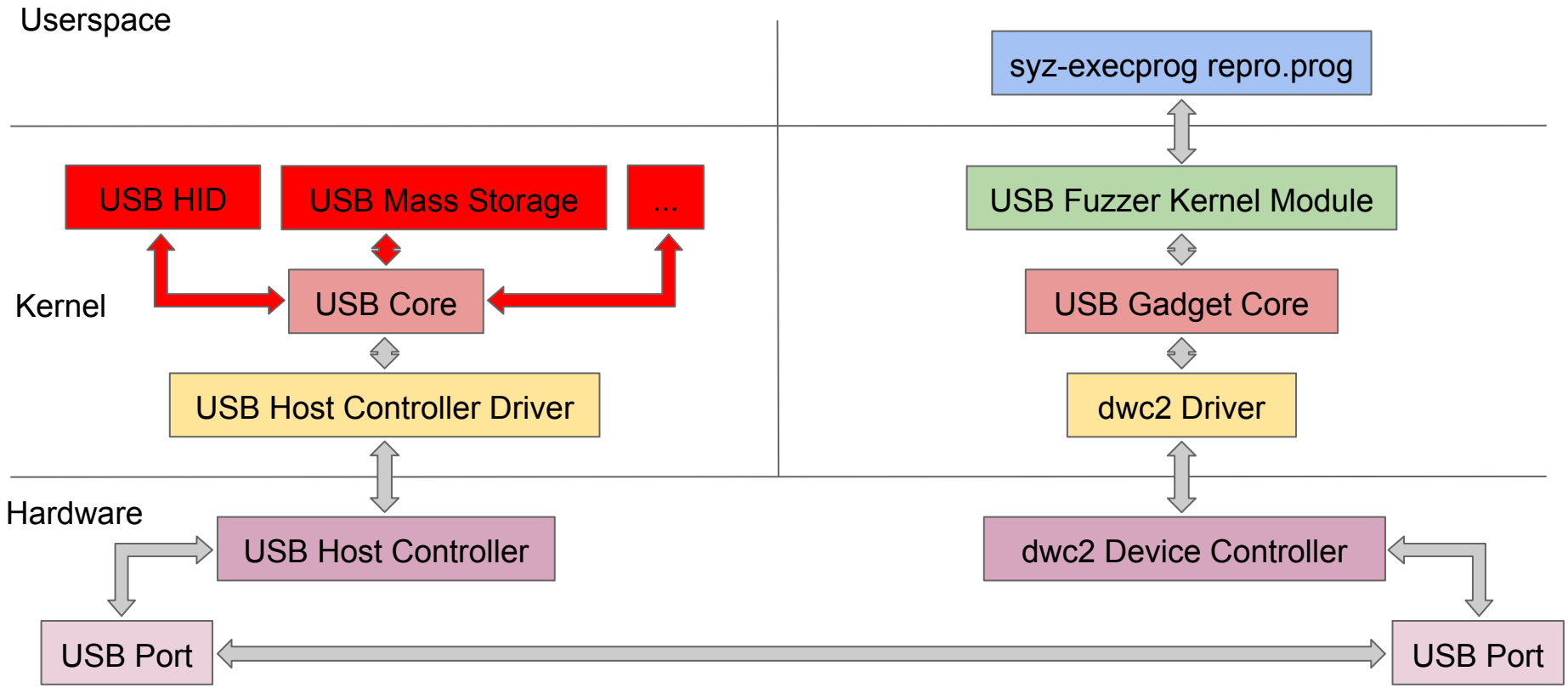
Let's Use the USB Fuzzer Module



And Plug RPi Zero into a Host



Running Reproducers via Raspberry Pi Zero



Demo: Crashing Linux Over USB

Bonus

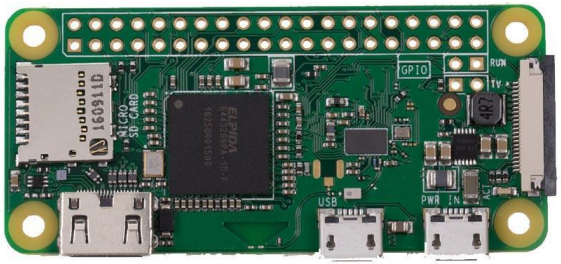
What We Have at This Point

- A corpus of syzkaller programs that describe USB devices
- A way to execute those programs over USB cable
- Let's fuzz a Windows host! :)

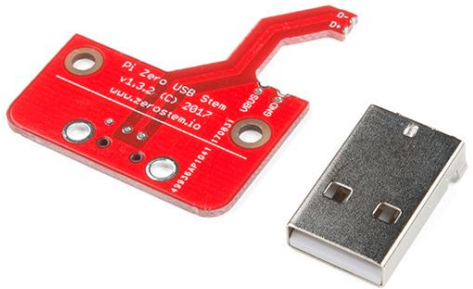
Getting Rid of the Wires

- Let's use a 10$ Raspberry Pi Zero W
- Has Wi-Fi chip on board
  - Can set up Wi-Fi hotspot + SSH server

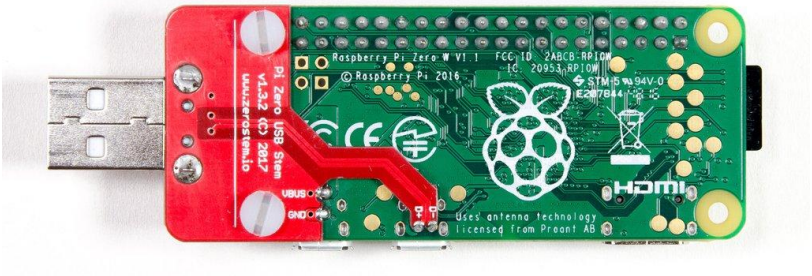
Raspberry Pi Zero W + Zero Stem



+



=



https://thepihut.com/products/raspberry-pi-zero-w

https://www.sparkfun.com/products/14526

https://shop.pimoroni.com/products/zero-stem-usb-otg-connector

Demo: Crashing Windows Over USB

Thanks!
Questions?

https://github.com/google/syzkaller/blob/usb-fuzzer/docs/linux/external_fuzzing_usb.md

https://syzkaller.appspot.com/

Andrey Konovalov <andreyknvl@google.com>

syzkaller@googlegroups.com